An Efficient Exploration Scheme for Datapath Width Optimization of Embedded Processor Systems

Mesbah, Uddin M.
Department of Computer Science and Communication Engineering, Graduate School of Information Science and Electrical Engineering, Kyushu University

Cao, Yun
Department of Computer Science and Communication Engineering, Graduate School of Information Science and Electrical Engineering, Kyushu University

Yasuura, Hiroto
Department of Computer Science and Communication Engineering, Graduate School of Information Science and Electrical Engineering, Kyushu University

出版情報：IEICE Technical Report, pp.10-16, 2002-03. 電子情報通信学会VLD研究会
バージョン：
権利関係：

An Efficient Exploration Scheme for Datapath Width Optimization of Embedded Processor Systems

Uddin M. Mesbah † Cao Yun † Hiroto Yasuura †

† Department of Computer Science and Communication Engineering
Graduate School of Information Science and Electrical Engineering
Kyushu University

6–1 Kasuga-koen, Kasuga-city, Fukuoka 816-8580 Japan
Tel. 092-583-7622

E-mail: {mesbah, cao, yasuura}@c.csce.kyushu-u.ac.jp

Abstract

Datapath width optimization is very effective for designing a custom-made processor system with low cost and less power/energy consumption. However, to determine an optimal value of datapath width, designers need to iteratively work on a number of customizations which results in a long design time. In order to reduce design time, we propose an efficient scheme for reducing the design exploration space for the optimization. Through a single-pass simulation for a reference customization and a model for estimating and evaluating performance, reduction in design exploration space can be achieved. Experimental results show that substantial reduction in design exploration space is possible.

Keywords:

Performance, cost, datapath width, optimization technique.

1 Introduction

Processor systems are used extensively in everyday use. Usually, such a system is application specific and has an increasing demand for low cost and less power/energy consumption while maintaining a minimum level of performance. Therefore, design optimization is quite important. However, the task of designing an optimized system often results in a long TAT (Turn Around Time). On the other hand, there is a growing *time-to-market* demand for these systems. Reducing the design time is therefore a crucial task.

Nevertheless, in designing processor systems, hardware of the processor is commonly considered as a collection of *hard-to-change* elements. Therefore, there is a common trend of viewing the hardware as a subject to IP (Intellectual Property) and the software is modified in order to support the hardware. On the contrary, our approach to obtain an optimized system is to fix the computational algorithm first and then tune the underlying hardware, while preserving the computational precision.

The datapath width of a processor system has a strong effect on the area, performance and energy consumption of the system [12] [11] [13]. Shackleford et al. have proposed a novel optimization, called datapath width optimization, which determines the optimal datapath width for such a system [1].

However, in order to view the processor system as a target of datapath width optimization, it is necessary to include a modifiable processor along with a high-level language and compiler as a part of the system. To provide such an environment, a customizable processor (Bung-DLX) [7], and a high level language, called Valen-C along with the retargetable Valen-C compiler [8] is developed.

Design methodologies have been proposed for determining the optimal datapath width using Bung-DLX and Valen-C [6] [1] [2]. Nevertheless, there is a design time bottelneck for such a methodology. The proposed methods result in a long TAT for designing a processor system.

The purpose of this work is to enhance support to shorten the TAT for such a design environment. We have observed that there is a time consuming iterative low-level simulation necessary for the proposed methodologies. In this paper, we propose an approach that reduces the number of such iterations. We introduce an approach for estimating the system performance at an early stage of design. Since, performance constraints are usually given in the the system specifications , it is possible to cut-off the repetitive simulation for those parameters that fail to satisfy the constraints.

The organization of this paper is as follows. Section 2 defines the terminologies and assumptions for this study. Section 3 describes related works and defines the problem. Section 4 presents the proposed approach. Section 5 shows experiment results. Section 6 summarizes this paper.

2 Preliminaries

2.1 Definitions

Processor System: We assume the following target processor system that is built on a single-chip, integrating an instruction memory(ROM), and a data memory(RAM) with a core processor.

Datapath Width: Datapath width is a popular measure of categorizing the processor and is equal to the width in bits of the internal datapath of the processor [12] [11] [13]. Alternatively, datapath width is the number of bits simultaneously transferred per memory access; it is the width of the processor components such as the arithmetic logic unit, the register file, or the bus. We assume that we can use arbitrary size of datapath.

Single Precision Operation: By single precision operation, we mean an operation whose reqired maximum width is not larger than the datapath width. For example, an addition of two 32 bit data is single precision operation on a 32-bit processor.

Multiple Precision Operation: By multiple precision operation, we mean an operation whose required width is larger than the datapath width. For example, an addition of two 20 bit data is double precision on a 16-bit processor.

Cycle Penalty Ratio: Cycle penalty occurs when an operation instance becomes multiple precision because the operation needs more execution cycles than the single precision case. For an operation instance with $exec_{sp}$ instruction(s) for single precesion and $exec_{pr}$ instructions for precision *pr*, we define the cycle penalty ratio as,

$$cpr_{pr} = \lceil \frac{exec_{pr}}{exec_{sp}} \rceil$$

If, for example, an operation executes 4 machine instructions for its single precision instance, and its cpr_2 is 2, then a double precision instance of the operation would require $2 \times 4 = 8$ machine instructions.

2.2 Basic System Requirements

For this study, we assume a cacheless, non-pipelined system integrating a core processor, instruction memory and data memory. ROM and RAM are used as instruction memory and data memory, respectively. In addition, we assume that:

- the computational precision of the application program can be determined,

- the parameters of the core processor is customizable,
- the performance constraints are given,
- cycle penalty ratios are known,
- the instruction word length remains invariant, and,
- the clock frequency of the core processor is fixed.

2.3 Soft-Core Processor and Valen-C

A soft-core-processor [7] is a customizable embedded processor having some design parameters. Some of the parameters are:

- datapath width,
- address width,
- number of registers, etc.

The designer can modify the parameters for each application, and then obtains a customized processor optimized for the application.

Valen-C [8] is an extension of C language [14]. In Valen-C, programmers can specify the required bit length of each variable in a program. For example, if two integer variables *x* and *y* require 12 and 20 bits respectively, the programmer can write

int12 x;
int20 y;

in the variable declaration. All the syntax and semantics of Valen-C is the same as C.

2.4 Criteria for Optimality

Criteria for optimality varies with design. Some applications may require small chip area while some other requiring low power/energy consuming circuitry. Our target is a complex system with processor and memories(RAM and ROM). Processor and memories show different area and power/energy characteristics on narrowing the datapath width or on modifying other parameters. Moreover, it might be impossible to sacrifice performance beyond an acceptance level. Therefore a *trade-off policy* considering the performance, area or power/energy consumed by the processor, memories and the whole system becomes necessary.

3 Datapath Width Optimization

When designing processor systems, it is sufficient to set the datapath width equal to the largest bit-width required by the application software. Having the datapath wider only results in extra size, but no increased performance. Previous works [5] [1] [2] [10] show that further reduction of datapath width can substantially reduce the area and energy required by the system. Although, there are efforts on how to gain advantage of the *knowledge of bitwidth*, a little has been addressed about improving the design methodology itself.

3.1 Evaluate-and-Redesign Approach for Optimization

The design flow of the known approach for datapath width optimization is shown in Figure 1.

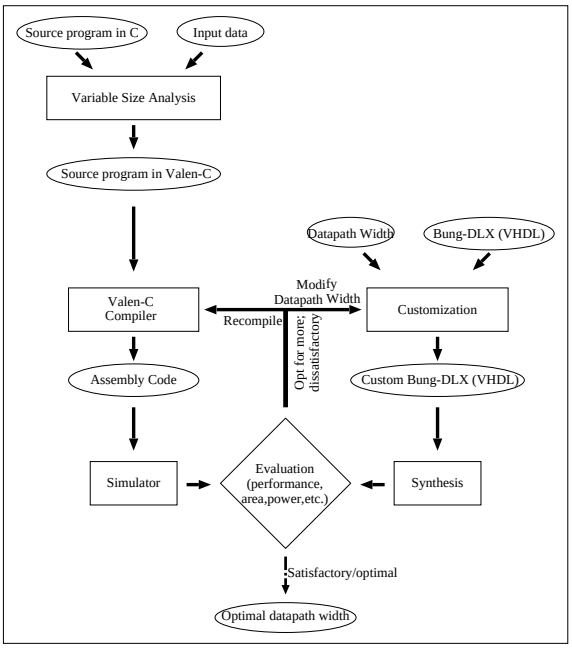

Figure 1. Design flow of the known approach

Initially the design begins with a soft-core-processor and the target application source program written in Valen-C. The design flow consists of four phases:

- Phase 1: The original “C” source program of the target application is rewritten in Valen-C. Variable size analysis is used in this phase.
- Phase 2: The soft-core-processor is customized for a datapath width.
- Phase 3: The source program in Valen-C is compiled for the customized processor.
- Phase 4: An evaluation (based on a *low level*, i.e., gate-level or lower , simulation and synthesis) gurantees whether the generated system satisfies design constraints. If the design constraints are not met or in order to find a suitable system, phase 2 to pase 4 are repeated for another datapath width in order to get a suitable system. However, where different customized systems should be compared for optimization, the repitition becomes trivial.

3.2 Problem Definition

The repitition of low-level simulations introduced in the known approach leads to a long TAT for designing an optimal system. Today, time-to-market design is very much necessary, and the demand confronts us with the following problem:

"How can datapath width be optimized efficiently(i.e.,speedily)?"

A speed-up is possible using a *high-level* estimation of system performance to reduce the design exploration space for datapath width optimization.

4 Reduction of Design Exploration Space

4.1 Basic Idea

Our proposal is:

"Reduce the design exploration space at an early stage of system design,"

since, it is generally less time consuming to find an optimal value from a reduced number of candidates. The reduction of design exploration space can be done by estimating and evaluating the performance of the system as a function of datapath width through a single-pass, high-level simulation.

A processor system is composed of: (1) the hardware components, and, (2) its software counterpart. Among the hardware components, there are the processor and the memories (data memory and instruction memory). The software part, on the other hand, consists of application programs which manipulates input data. Given a target processor along with a set of compiler specifications, the number of execution cycles is directly related to the code sequence of the application software. With an insight into the software and the way of its implementation on the hardware, it is possible to estimate the total number of execution cycles of the system through a one-pass, high-level simulation. The number of total execution cycles is directly related to system's performance.

As an illustration, let us consider the following code-sequence that will be executed on two different processors with datapath width of 32 bits and 21 bits, respectively.

```
int main()
{
    int16  x;
    int26  y;
    int30  z;
    z = x + y;
    . . . . . .
}
```

When the target processor has a datapath width of 32 bits, the Valen-C compiler is likely to convert the above program into the following machine code sequence:

```
load   x,R1;        R1 ← x15 ~ x0
load   y,R2;        R2 ← x25 ~ y0
add    R2,R1;       R1 ← R1 + R2
store  R1,z;        z29 ~ z0 ← R1
```

However, with the target processor having a datapath width of 21 bits, the code translates into:

```
load   x,R1;          R1 ← x15 ~ x0
load   y_lower,R2;    R2 ← y20 ~ y0
load   y_upper,R3;    R3 ← y25 ~ y21
add    R1,R2;         R2 ← R1 + R2
addc   #0,R3;         R3 ← R3 + carry
store  R2,z_lower;    z20 ~ z0 ← R2
store  R3,z_upper;    z29 ~ z21 ← R3
```

Now, let us focus on the original Valen-C program. For the assignment operation, we can easily determine the required bitwidth (*req_w*) for each of its operators, operands and results:

operator	req_w	operand	req_w	result	req_w
+	26	x	16	z	30
=	30	y	26		

For a 32-bit processor, all of the necessary operations are of single precision:

x: 1 single-precision load,
y: 1 single-precision load,
+: 1 single-precision add,
=: 1 single-precision store.

Assuming each single precision operation requires a single instruction, the result is a total of 4 machine instructions (i.e., execution cycles). However, for the 21-bit processor case, the necessary operations are:

x: 1 single-precision load (cpr=1),
y: 1 double-precision load(cpr=2),
+: 1 double-precision add (cpr=2),
=: 1 double-precision store (cpr=2).

For this case, the total number of execution cycles is 7.

Thus, it is possible to predict the total number of execution cycles even without any repetition of low-level simulations. The performance is determined as well.

4.2 Design Flow

The design flow of the proposed approach is shown in Figure 2. The flow consists of the following phases:

- Phase 1: The original "C" source program of the target application is rewritten in Valen-C. Variable size analysis is used in this phase.
- Phase 2: The system performance is estimated for an initial value of datapath width. This process consists of customizing the soft-core processor, compiling the application program for that processor, simulation, synthesis and so forth. Next, analysis for *access-count of variables*, and *precision and access*

count of functions is done. The system performance is then estimated for a number of datapath widths with an estimator function.

- Phase 3: Datapath widths that fail to satisfy the performance constraints are excluded. Thus, we get a set of *acceptable* datapath widths.
- Phase 4: The soft-core-processor is customized for an *acceptable* datapath width.
- Phase 5: The source program in Valen-C is compiled for the customized processor.
- Phase 6: An evaluation (based on a *low level*, i.e., gate-level or lower , simulation and synthesis) gurantees whether the generated system satisfies design constraints. If the design constraints are not met or in order to find a better system, phase 4 to pase 6 are repeated for another datapath width until all the customizations for the accepted datapath widths are *tested*. However, with a reduced number of design parameters obtained from phase 3, the number of repetitive simulations is reduced.

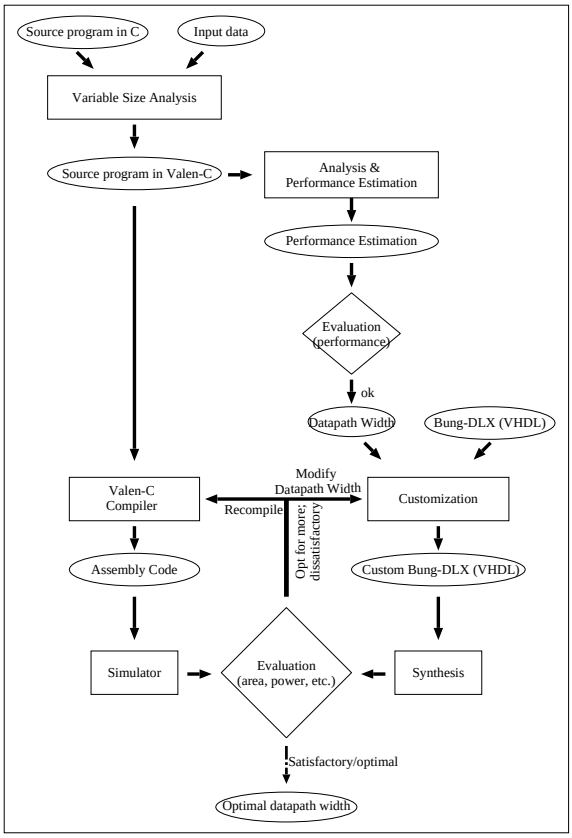

Figure 2. Design flow of the proposed approach

In this section we cover the main features of our proposed approach.

4.3 The Reference System

The reference system is assumed to have a datapath width of the largest variable in the program. However if the width exceeds the capability of existing technology, the widest possible supported-configuration is used instead. Initially, the performance of the *reference system* is estimated (Phase 2). Estimation process includes customizing the soft-core processor for the reference datapath width, compiling the application program for the customized processor, simulation and synthesis.

4.4 Analysis and Performance Estimation

The purpose of analysis phase is to take profile data for variable and operation instances. Analysis is done by executing the application program with a set of input data. Profile data includes the number each variable is accessed throughout the program run. It also includes the precision and the number of accesses to each operation (function) instances. Variables are divided into groups according to their effective bitwidth. Each function contains a subgroup, divided according to their precision on different instances.

4.4.1 Assumptions

The estimation model of our proposed approach utilizes the following assumptions:

- load operation is nesessary for each operand in an operation,
- store operation is necessary for each assignment operation, and,
- there is no register crowding.

4.4.2 Estimation Model

System performance is related to the number of execution cycles under typical inputs. In this study, the performance (P_{func}) for a given datapath width ω, is estimated by

$$P_{func}(\omega) = \frac{\beta}{\aleph_{exec\_cycle}(\omega)}$$

where β is a constant and $\aleph_{exec\_cycle}(\omega)$ is the total number of execution cycles for the application program necessary for memory accesses for that datapath width. A measure of $\aleph_{exec\_cycle}(\omega)$ as a function of datapath width is given by

$$\aleph_{exec\_cycle}(\omega) = \sum_{op_i} N_{exec\_cycle}(op_i, \omega)$$

where *op* is operation (*suffix i* is used to denote instance), $N_{exec\_cycle}(op_i, \omega)$ is the number of execution cycles for that operation instance op_i on a processor/system with datapath width ω and is determined by

$$N_{exec\_cycle}(op_i, \omega) = cpr_{pr}(op_i, \omega) \cdot n_{exec\_cycle}(op_{sp}, \omega)$$

where, *pr* is the precision of the operatiion instance, cpr is the cycle penalty ratio, and $n_{exec\_cycle}(op_{sp}, \omega)$ is the number of execution cycles for a single precision execution.

However, an operation described in a high-level language may need arithmetic/logic instruction(s), load/store instruction(s), or both. *op* stands for such an operation and should be understood from the context. If the operation instance involves only load/store instruction(s) (i.e., *op*=load/store), *pr* is determined by the size of the variable to be loaded(or, stored) and the datapath width ω. For any operation instance involving arithmetic/logic instruction(s) (i.e., *op*=function), *pr* is determined by the operation itself, the size of its operands (variables), and the datapath width ω. Nevertheless, it is likely that an operation would contain both kind of operations and a combination of the above is necessary.

For a datapath width equal to the size of the application program's largest variable, all the operations of that program will be single precision. If we narrow the datapath width, some of the operations would become multiple precision ones. This will increase the number of execution cycles for those operations. The net effect is an increase in the total number of execution cycles.

Now, with all the necessary information described above, it is possible to determine system performance as a function of datapath width.

4.4.3 First-Order Compensation

A first-order approximation compensates for the deviations of performance(P_{func}) introduced by our *profile-data*-based estimator function. The final estimated performance, $P_{est(\omega)}$, for a system with datapath width ω is approximated by

$$P_{est}(\omega) = \frac{P_{sim}(ref_\omega)}{P_{func}(ref_\omega)} \times P_{func}(\omega)$$

where, $P_{sim(ref_w)}$ is the performance for the reference system with datapath width ref_w, $P_{func(\omega)}$ is estimated performance for a system with datapath width ω, and, $P_{func(ref_\omega)}$is estimated performance for a system with datapath width ref_ω.

4.5 Evaluation and Reduction of Design Exploration Space

At this stage of design, (1) the performance constraints, and, (2) system performance as a function of datapath width are known. With this information, it is easy to detemine which of the systems satisfy the performance constraints. The datapath width of those satisfying the performance constraints are "accepted" datapath widths. Detailed design is done with these *accepted* values.

4.6 Detailed Design

Detailed design is done with the Evaluate-and-Redsign approach for a range of *accepted datapath width values*. Thus we cut reduce the design exploration space that would result into less number of explorations for datapath width optimization.

However, with an *accepted* value for datapath width, the core processor is customized and the application program is compiled for that processor. A simulation is used to determine the area, power consumption, etc. of the system. Again, an evaluation checks whether the customized system satisfies the design goals. In case the design goals is not met, a repetition is necessary as before. Since, we evaluate and redesign a reduced number of of targets, an acceleration is gained.

5 Experimental Results

In this section we will illustrate our experiment results. We used Bung-DLX as our target processor. Two applications are chosen for the experiment:

1. Lempel-Ziv algorithm, and,
2. Ccitt adpcm g721 encoder.

Both the programs are implemented in Valen-C. The system performance is assumed to lie within 5% of the reference case(32-bits for Lempel-Ziv, 19-bits for adpcm).

The strings used for the Lempel-Ziv program analysis are:

- binary file cat (a UNIX command),
- binary file pwd (a UNIX command),
- text file readme (a SAMBA utility program),
- text file readme.jis (a SAMBA utility program).

The range of possible datapath widths is 32-bits $\sim$ 8-bits (8, because of a restriction of the Valen-C compiler). The analysis & evaluation result is shown in Figure 3:

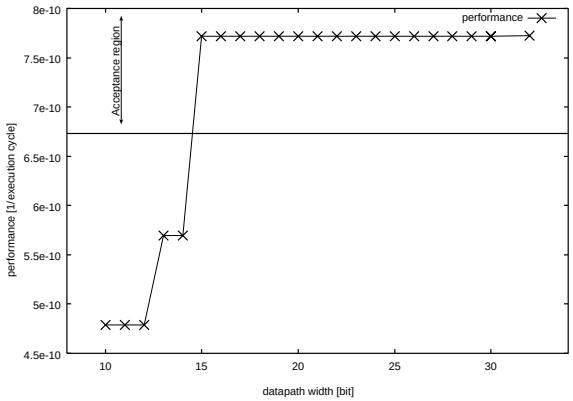

Figure 3. Performance estimation for the Lempel-Ziv program

From the Figure 3 we can easily verify that the acceptable rage of datapath width is 32-bits $\sim$ 15-bits. That is, the obtained reduction of design space is approximately 25%. We can now continue with these values and opt for the optimal datapath width. The criteria for optimality depends on the particularity of the system.

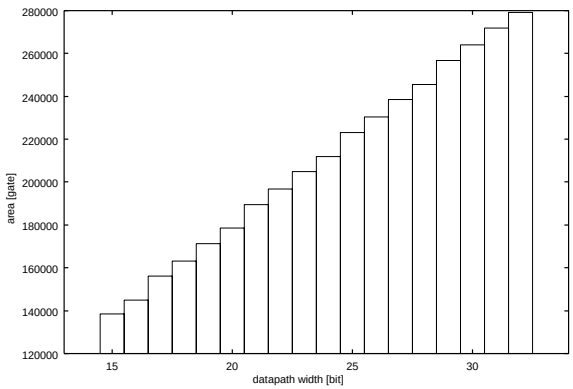

Figure 4. Area of the system for the Lempel-Ziv program

As an illustration, if area is the subject to concern, we determine the system area and see that the optimal datapath width is 15-bits (Figure 4).

For the adpcm encoder, standard input stream is used. The range of possible datapath widths is 19-bits $\sim$ 8-bits. The analysis result is shown in Figure 5.

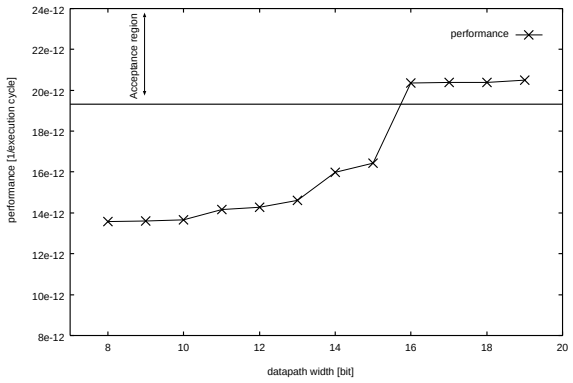

Figure 5. Performance estimation for the Ccitt adpcm encoder program

The range acceptable values of datapath width is 16-bits $\sim$ 19-bits. The design space is reduced to about 66%. The optimal datapath width for the minimum chip area is determined to be 16-bits.

5.1 Discussion

Our proposed approach reduces the number of repetition of the time consuming low-level simulations. However, the performance estimations of the proposed approach is compared with that of the existing approach (Figure 6, Figure 7).

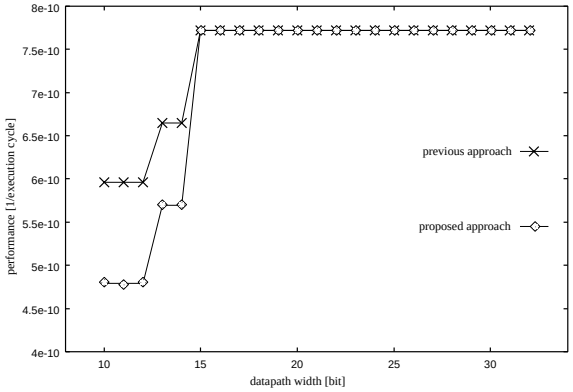

Figure 6. Comparison between the two approaches for the lempel-Ziv algorithm

Figure 6 illustrates that the proposed method introduces a negligible error. Further calculations show that the error rate lies below 20%, with a minimum rate of 0.03%. For datapath width equal to 15-bits or above, the error is below 0.06%, which is very substantial.

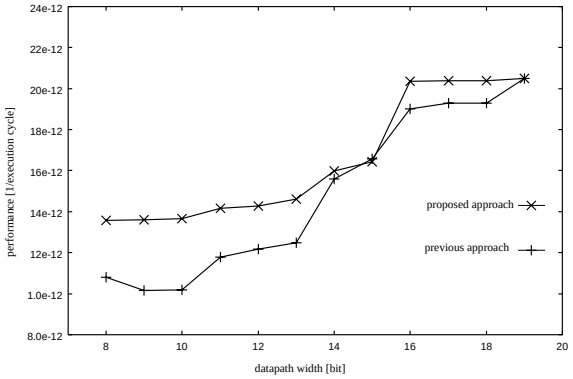

Figure 7. Comparison between the two approaches for the adpcm g721 encoder algorithm

However, for the adpcm encoder, the error rate rise up to 34%, at the narrower datapath widths (Figure 7). The error rate lies below 7% for datapth width 19-bits $\sim$ 14-bits.

Nevertheless, the significant feature of our approach is that the rise and falls in the performance curve resembles the graph of the other approach.

In the lower datapath width region, the error rate grows rapidly. As a trivial effect of datapath width narrowing, the number of multiple precision operations increases. In our simplistic model, the execution cycles due to *spill-instructions* (i.e., codes inserted by the compiler – such as *nop*s, or, variable declarations) was compensated by relating with the reference case. Improvement in the compensation model would reduce the error to a much

acceptable level. Another reason is the application program. A large number of variables in the Lempel-Ziv program lies near the 15-bit region. Below this region, the multiple-precision operations increase drastically. Yet, the increase of *nop*s or other codes inserted by the compiler is affected less. Similar is the case for the adpcm encoder program.

We conclude this discussion with the argument that our approach is quite effective as the estimated system performance shows a very similar behavior to the simulative performance and that a better compensation model may remedy the error rate.

6 Conclusion

In this paper, we have proposed an efficient approach which, through an early estimation and evaluation of system performance, reduces the design exploration space while determining the optimal datapath width. From our experience, this results in a reduction of design time although some overhead is introduced during the analysis. The analysis is done manually and is difficult to correlate with the conventional approach.

Currently, we are working on the automation of our proposed approach. We are also working on area,power/energy consumption concerns to integrate in our analysis. Further enhancement like considering the codes inserted by the compiler, or, covering a pipelined system with cache memory is necessary for real world system. Those remain as our future interest.

References

[1] B. Shackleford, M. Yasuda, E. Okushi, H. Koizumi, H. Tomiyama, A. Inoue, and H. Yasuura, "Embedded system cost optimization via datapath width adjustment," IEICE Trans. Inf. & Syst., vol. E80-D, no. 10, pp. 974-981, Oct. 1997.

[2] B. Shackleford, M. Yasuda, E. Okushi, H. Koizumi, H. Tomiyama, and H. Yasuura, "Memory-CPU Size optimization for Embedded system Designs," Proc. of 34th Design Automation Conference (34th DAC), pp. 246-251, June 1997.

[3] D.E. Thomas, J.K. Adams, and H. Schmit. "A Model and Methodology for Hardware-Software Codesign," IEEE Design& Test of Computers, Vol. 10, no. 3, pp. 6-15, Sept. 1993.

[4] R. K. Gupta, C. N. Coelho, Jr., and G. De Micheli, "Program implementation schemes for hardware-software systems," IEEE Computer, Vol. 27, no. 1, pp. 48-55, Jan. 1994.

[5] Scott Mahlke, *et al.*,"Bitwidth Cognizant Architecture Synthesis of Custom Hardware Accelerators," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Synthesis, Vol. 20, no. 11, pp. 1355-1371, November 2001.

[6] H. Yasuura, H. Tomiyama, A. Inoue, and F. N. Eko, "Embedded System Design Using Soft-Core Processor and Valen-C," Journal of Information Science and Engineering, No. 14, pp. 587-603, August 1998.

[7] F. N. Eko, A. Inoue, H. Tomiyama, and H. Yasuura, "Soft-Core Processor Architecture for Embedded System Design," IEICE Trans. on Electronics, Vol. E81-C No. 9, pp. 1416-1423, Sep. 1998.

[8] A.Inoue, H. Tomiyama, T. Okuma, H. Kanbara, and H. Yasuura, "Language and Compiler for Optimizing Datapath Width of Embedded Systems," IEICE Trans. Fundamentals, Vol. E81-A, No. 12, pp. 2595-2604, Dec. 1998.

[9] H. Yamashita, H. Tomiyama, A. Inoue, F. N. Eko, T. Okuma, and H. Yasuura, "Variable Size Analysis for Datapath Width Optimization," Proc. of Asia Pacific Conference on Hardware Description Languages(APCHDL '98), pp. 69-74, July 1998.

[10] Y. Cao, H. Yasuura, "A System-level Energy Minimization Approach Using Datapath Width Optimization."

[11] D. A. Patterson and J. L. Hennessy, "Computer Organization: The Hardware/Software Interface," 2nd edition, Morgan Kaufmann Publishers, Inc., 1997.

[12] J. L. Hennessy and D. A. Patterson, "Computer Architecture: A Quantitative Approach," 2nd edition, Morgan Kaufmann Publishers, Inc., 1996.

[13] Anthony Ralston, Edwin D. Reilly, David Hemmendinger, "Encyclopedia of Computer Science," 4th edition, Nature Publishing Group, 2000.

[14] Brian W. Kernighan, Dennis M. Ritchie, "The C Programming Language," 2nd edition, Prentice-Hall, Inc., 1988.